RTA-OSEK

Binding Manual: HC12X/COSMIC

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

Tel.:+49 (711) 8 96 61-102
Fax:+49 (711) 8 96 61-106

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

Tel.: +1 (888) ETAS INC
Fax: +1 (734) 997-94 49

www.etasinc.com

Japan

ETAS K.K.
Queen's Tower C-17F,
2-3-5, Minatomirai, Nishi-ku,
Yokohama, Kanagawa
220-6217 Japan

Tel.: +81 (45) 222-0900
Fax: +81 (45) 222-0956

www.etas.co.jp

France

ETAS S.A.S.
1, place des États-Unis
SILIC 307
94588 Rungis Cedex

Tel.: +33 (1) 56 70 00 50
Fax: +33 (1) 56 70 00 51

www.etas.fr

Korea

ETAS Korea Co. Ltd.
3F, Samseung Bldg. 61-1
Yangjae-dong, Seocho-gu
Seoul

Tel.: +82 (2) 57 47-016
Fax: +82 (2) 57 47-120

www.etas.co.kr

Great Britain

ETAS UK Ltd.
Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ

Tel.: +44 (0) 1283 - 54 65 12
Fax: +44 (0) 1283 - 54 87 67

www.etas-uk.net

Copyright Notice

Version: RM00064-002

Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on any part of LiveDevices. While the information contained herein is assumed to be accurate, LiveDevices assumes no responsibility for any errors or omissions.

In no event shall LiveDevices, its employees, its contractors or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees or expenses of any nature or kind.

Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

All other product names are trademarks or registered trademarks of their respective owners.

Contents

1 About this Guide

This guide provides port specific information for the HC12X/COSMIC implementation of LiveDevices' RTA-OSEK.

A port is defined as a specific target microcontroller/target toolchain pairing. This guide tells you about integration issues with your target toolchain and issues that you need to be aware of when using RTA-OSEK on your target hardware. Port specific parameters of implementation are also provided, giving the RAM and ROM requirements for each object in the RTA-OSEK Component and execution times for each API call to the RTA-OSEK Component.

1.1 Who Should Read this Guide?

It is assumed that you are a developer. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

1.2 Conventions

Important: Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

Portability: Notes that appear like this describe things that you will need to know if you want to write code that will work on any processor running the RTA-OSEK Component.

In this guide you'll see that program code, header file names, C type names, C functions and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Toolchain Issues

In this chapter, you'll see the important details that you need to know about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware *and* the compiler toolchain. You must make sure that you build your application with this toolchain.

If you are interested in using a different version of the same toolchain, you should contact LiveDevices to confirm whether or not this is possible.

2.1 Memory Model

The HC12 architecture supports the use of one-byte addresses for the first 256 bytes of the address space, called the "zero page" section or "zpage". Conventionally, however, low memory addresses are used for I/O flags. The RTA-OSEK Component therefore makes no special use of zpage, and the modifier `@dir` is not used.

The HC12 architecture can also support three kinds of memory banking:

- a 3-byte CALL/RTC mechanism in which 16k banks of code are mapped into and out of the space between 0x8000 and 0xBFFF. The PPAGE register stores the index of the currently-mapped bank.
- a banked 4k data space at 0x7000-0x7FFF selectable by writing to the DPAGE register.
- a banked 1k data space in low memory selectable by writing to the EPAGE register.

Only the first of these is explicitly catered for by RTA-OSEK. All library code has been compiled without the `@far` modifier, and must therefore appear unbanked address space. If banked code is used for application code, most of what is required is the user's responsibility. Where the API calls `TerminateTask()` and `WaitEvent()` are used in application code, if a bank switch is required the RTA-OSEK Component makes the necessary modification to PPAGE.

In order to support this, the user is required to make known the location of PPAGE when linking applications, using a linker command of the form

```
+def os_ppage=0x30
```

or whatever the location of PPAGE is on that target. If PPAGE does not exist, the location of any unused byte in RAM should be given.

No support is provided for the use of DPAGE and EPAGE, so these may only be used in ways which do not affect the RTA-OSEK Component in any way.

2.2 Compiler

The RTA-OSEK Component was built using the following compiler:

Vendor	Cosmic
Compiler	cxs12x
Version	V4.6a

The compulsory compiler options for application code are shown in the following table:

Option	Description
`+nowiden`	Do not widen char parameters to integers

The C file that RTA-OSEK generates from your OIL configuration file is called `osekdefs.c`. This file defines configuration parameters for the RTA-OSEK Component when running your application.

The compulsory compiler options for `osekdefs.c` are shown in the following table:

Option	Description
`+nowiden`	Do not widen char parameters to integers

2.3 Assembler

The RTA-OSEK Component was built using the following assembler:

Vendor	Cosmic
Assembler	cxs12x
Version	V4.6a

The assembly file that RTA-OSEK generates from your OIL configuration file is called `osgen.s`. This file defines configuration parameters for the RTA-OSEK Component when running your application.

2.4 Linker/Locator

The compulsory linker/locator options for an RTA-OSEK application are shown in the following table:

Option	Description
`+def os_ppage=<value>`	<value>=the address of PPAGE if any, or otherwise any unused RAM address

In addition to the sections used by application code, the following RTA-OSEK sections must be located:

Sections	Rom/Ram	Description
os_pid	ROM	RTA-OSEK read-only data
os_pird	ROM	RTA-OSEK initialization data
os_vectbl	ROM	Relocatable vector table if generated by RTA-OSEK GUI
os_vectbl1	ROM	Fixed vector table if generated by RTA-OSEK GUI
os_pir	RAM	RTA-OSEK initialized data
os_pur	RAM	RTA-OSEK uninitialized data

All Cosmic run-time libraries are compatible with RTA-OSEK.

2.5 Debugger

ORTI is the OSEK Run-Time Interface that is supported by RTA-OSEK. Support is provided for the debuggers in the following table. Further information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*.

ORTI compatible debuggers	iSYSTEM winIDEA

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *CPU12X Reference Manual*.

The following table shows how RTA-OSEK IPLs relate to interrupt priorities on the target hardware:

IPL Value	High Byte Of Condition Code Register	I Bit In Condition Code Register	Description
0	0	0	User level
1	1	0	Category 1 and 2 interrupts
2	2	0	Category 1 and 2 interrupts
3	3	0	Category 1 and 2 interrupts
4	4	0	Category 1 and 2 interrupts
5	5	0	Category 1 and 2 interrupts
6	6	0	Category 1 and 2 interrupts
7	7	0	Category 1 and 2 interrupts
8	any value	1	Category 1 interrupts only

3.1.2 Interrupt Vectors

For the allocation of Category 1 and Category 2 interrupt handlers to interrupt vectors on your target hardware, the following restrictions apply:

Vector	Legality
0xFFFE	RESET cannot be used
0x10, 0xF4 to 0xF8, 0xFFFA, 0xFFFC	These vectors used by non-maskable interrupts can only be defined as Category 1 interrupts with a priority of 8.
0x12 to 0xF2	These vectors can be defined is Category 1 or Category 2 and must have a priority between 1 and 7.

The HC12X has some vectors that have a fixed location (vectors 0xFFFC, 0xFFFA) and the rest of the vectors are relocatable. If a vector table is generated it has two sections. Section `os_vectbl1` contains the fixed location vectors and must be located at 0xfffa. Section `os_vectbl` contains the relocatable vectors and can be placed at any of the locations outlined in the processor documentation. The user is responsible for initializing the Interrupt Vector Base Register (IVBR).

3.1.3 Interrupt Priority Levels

The priority at which a hardware interrupt is taken is set in the INT_CFDATA registers under the control of the INT_CFADDR register.

The RTA-OSEK GUI generates a table, called `os_InitIrqLevels`, which must be used to initialize the INT_CFDATA registers. This table contains the priority levels for interrupts defined in the application.

Important: The `os_InitIrqLevels` table must be copied to the INT_CFDATA registers before the call to `StartOS()` otherwise interrupts will not work correctly.

The `init_target()` function in `target.c` in the example application (located in `<RTA-OSEK install directory>\COS12X\Example\`) gives an example of how to copy `os_InitIrqLevels` to the correct location.

3.1.4 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Cosmic C compiler can generate appropriate interrupt handling code for a C function decorated with the `@interrupt` function qualifier. You can find out more in your compiler documentation.

3.1.5 Category 2 Handlers

Category 2 ISRs are provided with a C function context by the RTA-OSEK Component, since the RTA-OSEK Component handles the interrupt context

itself. The handlers are written using the OSEK OS standard `ISR()` macro, shown in Code Example 3:1.

```
#include "MyISR.h"
ISR(MyISR) {
  /* Handler routine */
}
```

Code Example 3:1 - Category 2 ISR Interrupt Handler

You must not insert a return from interrupt instruction in such a function. The return is handled automatically by the RTA-OSEK Component.

3.1.6 Vector Table Issues

When you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated within `osgen.s`.

Note that a generated vector table omits the reset vector entry. If you choose to provide your own vector table, it must contain an entry for each interrupt handler, including the Category 2 interrupt handlers in RTA-OSEK.

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers (VV represents the 2 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
0xVV	`_os_wrapper_VV`
e.g. 0x90	`_os_wrapper_90`

3.2 Register Settings

The RTA-OSEK Component does not require the initialization of registers before calling `StartOS()`.

The RTA-OSEK Component uses the following hardware registers. They should not be altered by user code.

Registers Used	Notes
High byte of CCR	The high byte of the CCR holds the current priority level
I bit in the CCR	The I bit enables and disables interrupts.

3.3 Stack Usage

3.3.1 Number of Stacks

A single stack is used. The first argument to `StackFaultHook` is always 0.

`StackOffsetType` is a scalar, representing the number of bytes on the stack, with C type: `unsigned short`

3.3.2 Stack Usage within API Calls

The maximum stack usage within RTA-OSEK API calls, excluding calls to hooks and callbacks, is as follows:

Standard

API max usage (bytes): 15

Timing

API max usage (bytes): 15

Extended

API max usage (bytes): 23

To determine the correct stack usage for tasks that use other library code, you may need to contact the vendor to find out more about library call stack usage.

4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OSEK Component.

The RTA-OSEK Component is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give six classes of RTA-OSEK, depending on whether your application uses events, shared task priorities and/or multiple (queued) task activations. You should identify which class your application belongs to and then use the figures from the appropriate column in the table.

The following hardware was used to take the measurements in this chapter:

Processor	**Mp9s12xdp512**
Clock speed (MHz)	40
Code memory	On-chip FLASH
Read-only data memory	On-chip FLASH
Read-write data memory	On-chip RAM

4.1 Functionality

The OSEK Operating System Specification specifies four conformance classes. These attributes apply to *systems* built with OSEK OS objects. The following table specifies the number of OSEK OS and COM objects supported per conformance class.

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	16	16	16	16	16	16
Maximum number of not suspended tasks	16	16	16	16	16	16
Maximum number of priorities	16	16	16	16	16	16
Number of tasks per priority (for BCC2 and ECC2)	n/a	16	16	n/a	16	16
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	16	16	16
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255
Limits for the number of application modes	255					

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

The following tables give the ROM and RAM overheads for the RTA-OSEK Component (in bytes). The OSEK COM overheads are quoted separately. If you do not use messages, your application will not include this overhead for the parts of OSEK COM required to implement messaging.

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	12	12	12	12	12	12
	ROM	92	92	92	92	92	92
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	22	22	22	22	22	22
	ROM	127	127	127	127	127	127
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	28	28	28	28	28	28
	ROM	147	147	147	147	147	147
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. RTA-OSEK provides additional sub-task types for each task type in OSEK (basic and extended), determined by the offline configuration tools. They are as follows:

OSEK Class	Termination	Arithmetic
BCC1	Lightweight	Integer or Floating-Point
BCC1	Heavyweight	Integer or Floating-Point
BCC2	Light or Heavy	Integer or Floating-Point
ECC1	Heavyweight	Integer
ECC1	Heavyweight	Floating-Point
ECC2	Heavyweight	Integer
ECC2	Heavyweight	Floating-Point

The following tables give the ROM and/or RAM requirements (in bytes) for each OS object in the RTA-OSEK Component. (Note that the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. A default message of size 10 bytes was used for both CCCA and CCCB. The CCCB message size includes queued messages.)

Standard

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	17	17	17	17	17	17
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	19	19	19	19	19	19
BCC2 task	RAM	n/a	3	5	n/a	3	5
	ROM	n/a	20	24	n/a	20	24
ECC1, Integer task	RAM	n/a	n/a	n/a	11	11	11
	ROM	n/a	n/a	n/a	29	29	29
ECC1, floating-point task	RAM	n/a	n/a	n/a	12	12	12
	ROM	n/a	n/a	n/a	29	29	29
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	13
	ROM	n/a	n/a	n/a	n/a	n/a	33
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	14
	ROM	n/a	n/a	n/a	n/a	n/a	33
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	25	25	25	25	25	25
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	35	35	35	35	35	35
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	5	5	5	5	5	5
	ROM	29	29	29	29	29	29
Counter	RAM	2	2	2	2	2	2
	ROM	11	11	11	11	11	11
Message	RAM	11	11	11	21	21	21
	ROM	10	10	10	27	27	27
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	7	7	7	7	7	7
	ROM	16	16	16	16	16	16
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	5	5	5	5	5	5
	ROM	22	22	22	22	22	22
BCC1 Heavyweight task	RAM	7	7	7	7	7	7
	ROM	24	24	24	24	24	24
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	25	29	n/a	25	29
ECC1, Integer task	RAM	n/a	n/a	n/a	16	16	16
	ROM	n/a	n/a	n/a	34	34	34
ECC1, floating-point task	RAM	n/a	n/a	n/a	17	17	17
	ROM	n/a	n/a	n/a	34	34	34
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	18
	ROM	n/a	n/a	n/a	n/a	n/a	38
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	19
	ROM	n/a	n/a	n/a	n/a	n/a	38

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	5	5	5	5	5	5
	ROM	46	46	46	46	46	46
Category 2 ISR, floating-point	RAM	6	6	6	6	6	6
	ROM	64	64	64	64	64	64
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	5	5	5	5	5	5
	ROM	29	29	29	29	29	29
Counter	RAM	2	2	2	2	2	2
	ROM	11	11	11	11	11	11
Message	RAM	11	11	11	21	21	21
	ROM	10	10	10	27	27	27
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	7	7	7	7	7	7
	ROM	16	16	16	16	16	16
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Extended

Configuration Events Shared Task Priorities Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	6	6	6	6	6	6
	ROM	26	26	26	26	26	26
BCC1 Heavyweight task	RAM	8	8	8	8	8	8
	ROM	26	26	26	26	26	26
BCC2 task	RAM	n/a	9	11	n/a	9	11
	ROM	n/a	27	31	n/a	27	31
ECC1, Integer task	RAM	n/a	n/a	n/a	17	17	17
	ROM	n/a	n/a	n/a	36	36	36
ECC1, floating-point task	RAM	n/a	n/a	n/a	18	18	18
	ROM	n/a	n/a	n/a	36	36	36
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	19
	ROM	n/a	n/a	n/a	n/a	n/a	40
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	20
	ROM	n/a	n/a	n/a	n/a	n/a	40
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	50	50	50	50	50	50
Category 2 ISR, floating-point	RAM	7	7	7	7	7	7
	ROM	68	68	68	68	68	68
Resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Alarm	RAM	5	5	5	5	5	5
	ROM	31	31	31	31	31	31
Counter	RAM	2	2	2	2	2	2
	ROM	13	13	13	13	13	13
Message	RAM	11	11	11	21	21	21
	ROM	12	12	12	29	29	29
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (writable)	RAM	10	10	10	10	10	10
	ROM	10	10	10	10	10	10
Schedule	RAM	9	9	9	9	9	9
	ROM	20	20	20	20	20	20
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

4.2.3 Size of Linkable Modules

The RTA-OSEK Component is demand linked. This means that each API call is placed into a separately linkable module. The following sections list the module sizes (in bytes) for each API call in the 3 RTA-OSEK build types (standard, timing, and extended).

In some cases there are multiple variants of particular API calls. This is because the offline configuration of RTA-OSEK can determine when optimized versions of the API calls can be used. The smallest and fastest call will be selected. In these cases, modules sizes are given for each variant under the particular configuration of the RTA-OSEK Component for which the call is valid.

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.

Variant	Description
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	81	130	173	87	138	192
	NS		69	118	160	75	126	180
	KL	2	45	92	136	49	96	157

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	16	16	16	16	16	16
ChainTask	SWL	1, 8	68	117	162	74	125	180
	SWH	1, 9	95	142	183	101	150	202
	NSL	8	68	117	162	74	125	180
	NSH	9	89	136	180	95	144	196
Schedule			57	57	76	57	57	76
GetTaskID			23	23	23	23	23	23
GetTaskState			60	60	60	72	72	72
EnableAllInterrupts			8	8	8	8	8	8
DisableAllInterrupts			21	21	21	21	21	21
ResumeAllInterrupts			13	13	13	13	13	13
SuspendAllInterrupts			30	30	30	30	30	30
ResumeOSInterrupts			13	13	13	13	13	13
SuspendOSInterrupts			37	37	37	37	37	37
GetResource	Task	7	27	27	30	27	27	30
	Combined	6	67	67	67	67	67	67
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	48	48	48	48	48	48
	Combined	6	90	90	90	90	90	90
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	167
	NS		n/a	n/a	n/a	78	78	155
	NS1i	10	n/a	n/a	n/a	37	n/a	n/a
	KL	2	n/a	n/a	n/a	57	57	135
	KL1i	2, 10	n/a	n/a	n/a	14	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	171	171	333
	fp	11	n/a	n/a	n/a	193	193	383
	1i	10	n/a	n/a	n/a	15	n/a	n/a
GetAlarmBase			45	45	45	45	45	45
GetAlarm			75	75	75	75	75	75
SetRelAlarm			101	101	101	101	101	101
SetAbsAlarm			125	125	125	125	125	125
CancelAlarm			55	55	55	55	55	55
InitCounter			48	48	48	48	48	48

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			50	50	50	50	50	50
osek_tick_alarm	<default>		56	56	56	56	56	56
	KL	2	34	34	34	34	34	34
osek_incr_counter			32	32	32	32	32	32
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			92	92	92	92	92	92
ShutdownOS	NoHook	12	18	18	18	18	18	18
	Hook	13	24	24	24	24	24	24
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			16	16	16	16	16	16
StopCOM			18	18	18	18	18	18
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	49	49	49	158	158	158
	CCCB	15	158	158	158	158	158	158
GetMessageResource			22	22	22	22	22	22
ReleaseMessageResource			20	20	20	20	20	20
GetMessageStatus			48	48	48	48	48	48
SendMessage	SW CCCA	1, 14	68	68	68	191	191	191
	SW CCCB	1, 15	170	170	170	191	191	191
	NS CCCA	14	68	68	68	191	191	191
	NS CCCB	15	170	170	170	191	191	191
	KL CCCA	2, 14	51	51	51	178	178	178
	KL CCCB	2, 15	157	157	157	178	178	178
main_dispatch	NoHook	12	126	126	155	126	126	155
	Hook	13	149	149	180	149	149	180
sub_dispatch	B1LF	19	14	14	14	14	14	14
	B1HI	20	56	56	56	56	56	56
	B1HF	21	62	62	62	62	62	62
	B2LI	22	n/a	39	66	n/a	39	66
	B2LF	23	n/a	45	72	n/a	45	72
	B2HI	24	n/a	94	162	n/a	94	162
	B2HF	25	n/a	100	168	n/a	100	168
	E1HI	26	n/a	n/a	n/a	243	243	319
	E1HF	27	n/a	n/a	n/a	249	249	325
	E2HI	28	n/a	n/a	n/a	n/a	n/a	319

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	325
ErrorHook support		16	19	19	19	19	19	19
	ServiceID	17	24	24	24	24	24	24
	Parameters	18	41	41	41	41	41	41
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	n/a	n/a	n/a	n/a	n/a	n/a
Timing_termination		4	n/a	n/a	n/a	n/a	n/a	n/a
ActivateTaskset	SW	1	48	89	137	54	105	155
	NS		36	77	125	42	93	143
	KL	2	13	58	106	19	74	124
ChainTaskset	SWL	1, 8	34	77	125	34	87	137
	SWH	1, 9	66	118	166	66	128	178
	NSL	8	34	77	125	34	87	137
	NSH	9	60	112	160	60	122	172
GetTasksetRef			10	10	10	10	10	10
MergeTaskset			36	36	36	36	36	36
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			38	38	38	38	38	38
TestSubTaskset			47	47	47	47	47	47
TestEquivalentTaskset			44	44	44	44	44	44
TickSchedule	SW	1	108	102	102	102	102	102
	NS		93	87	87	87	87	87
	KL	2	76	70	70	70	70	70
AdvanceSchedule	SW	1	97	93	93	93	93	93
	NS		82	78	78	78	78	78
	KL	2	63	57	57	57	57	57
StartSchedule			57	57	57	57	57	57
StopSchedule			39	39	39	39	39	39
GetScheduleStatus			69	69	69	69	69	69
GetScheduleValue			49	49	49	49	49	49
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			10	10	10	10	10	10
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			10	10	10	10	10	10

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestArrivalpointWritable			21	21	21	21	21	21
GetExecutionTime			3	3	3	3	3	3
GetLargestExecutionTime			7	7	7	7	7	7
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			21	21	21	21	21	21
Memory copy			15	15	15	15	15	15
Context save/restore			30	30	30	30	30	30
Floating point support			12	12	12	12	12	12
Interrupt support			23	23	23	23	23	23

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	81	130	173	87	138	192
	NS		69	118	160	75	126	180
	KL	2	45	92	136	49	96	157
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	16	16	16	16	16	16
ChainTask	SWL	1, 8	68	117	162	74	125	180
	SWH	1, 9	95	142	183	101	150	202
	NSL	8	68	117	162	74	125	180
	NSH	9	89	136	180	95	144	196
Schedule			66	66	85	66	66	85
GetTaskID			23	23	23	23	23	23
GetTaskState			60	60	60	72	72	72
EnableAllInterrupts			8	8	8	8	8	8
DisableAllInterrupts			21	21	21	21	21	21
ResumeAllInterrupts			13	13	13	13	13	13
SuspendAllInterrupts			30	30	30	30	30	30
ResumeOSInterrupts			13	13	13	13	13	13
SuspendOSInterrupts			37	37	37	37	37	37
GetResource	Task	7	27	27	30	27	27	30

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Combined	6	67	67	67	67	67	67
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	57	57	57	57	57	57
	Combined	6	108	108	108	108	108	108
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	167
	NS		n/a	n/a	n/a	78	78	155
	NS1i	10	n/a	n/a	n/a	37	n/a	n/a
	KL	2	n/a	n/a	n/a	57	57	135
	KL1i	2, 10	n/a	n/a	n/a	14	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	201	201	364
	fp	11	n/a	n/a	n/a	223	223	414
	1i	10	n/a	n/a	n/a	45	n/a	n/a
GetAlarmBase			45	45	45	45	45	45
GetAlarm			75	75	75	75	75	75
SetRelAlarm			101	101	101	101	101	101
SetAbsAlarm			125	125	125	125	125	125
CancelAlarm			55	55	55	55	55	55
InitCounter			48	48	48	48	48	48
GetCounterValue			50	50	50	50	50	50
osek_tick_alarm	<default>		56	56	56	56	56	56
	KL	2	34	34	34	34	34	34
osek_incr_counter			32	32	32	32	32	32
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			115	115	115	115	115	115
ShutdownOS	NoHook	12	18	18	18	18	18	18
	Hook	13	24	24	24	24	24	24
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			16	16	16	16	16	16
StopCOM			18	18	18	18	18	18
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	49	49	49	158	158	158
	CCCB	15	158	158	158	158	158	158

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageResource			22	22	22	22	22	22
ReleaseMessageResource			20	20	20	20	20	20
GetMessageStatus			48	48	48	48	48	48
SendMessage	SW CCCA	1, 14	68	68	68	191	191	191
	SW CCCB	1, 15	170	170	170	191	191	191
	NS CCCA	14	68	68	68	191	191	191
	NS CCCB	15	170	170	170	191	191	191
	KL CCCA	2, 14	51	51	51	178	178	178
	KL CCCB	2, 15	157	157	157	178	178	178
main_dispatch	NoHook	12	161	161	190	161	161	190
	Hook	13	186	186	215	186	186	215
sub_dispatch	B1LF	19	10	10	10	10	10	10
	B1HI	20	57	57	57	57	57	57
	B1HF	21	63	63	63	63	63	63
	B2LI	22	n/a	28	55	n/a	28	55
	B2LF	23	n/a	34	61	n/a	34	61
	B2HI	24	n/a	81	149	n/a	81	149
	B2HF	25	n/a	87	155	n/a	87	155
	E1HI	26	n/a	n/a	n/a	256	256	330
	E1HF	27	n/a	n/a	n/a	262	262	336
	E2HI	28	n/a	n/a	n/a	n/a	n/a	330
	E2HF	29	n/a	n/a	n/a	n/a	n/a	336
ErrorHook support		16	19	19	19	19	19	19
	ServiceID	17	24	24	24	24	24	24
	Parameters	18	41	41	41	41	41	41
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	34	34	34	34	34	34
Timing_termination		4	49	49	49	49	49	49
ActivateTaskset	SW	1	48	89	137	54	105	155
	NS		36	77	125	42	93	143
	KL	2	13	58	106	19	74	124
ChainTaskset	SWL	1, 8	34	77	125	34	87	137
	SWH	1, 9	66	118	166	66	128	178
	NSL	8	34	77	125	34	87	137
	NSH	9	60	112	160	60	122	172
GetTasksetRef			10	10	10	10	10	10
MergeTaskset			36	36	36	36	36	36

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			38	38	38	38	38	38
TestSubTaskset			47	47	47	47	47	47
TestEquivalentTaskset			44	44	44	44	44	44
TickSchedule	SW	1	108	102	102	102	102	102
	NS		93	87	87	87	87	87
	KL	2	76	70	70	70	70	70
AdvanceSchedule	SW	1	97	93	93	93	93	93
	NS		82	78	78	78	78	78
	KL	2	63	57	57	57	57	57
StartSchedule			57	57	57	57	57	57
StopSchedule			39	39	39	39	39	39
GetScheduleStatus			69	69	69	69	69	69
GetScheduleValue			49	49	49	49	49	49
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			10	10	10	10	10	10
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			10	10	10	10	10	10
TestArrivalpointWritable			21	21	21	21	21	21
GetExecutionTime			56	56	56	56	56	56
GetLargestExecutionTime			14	14	14	14	14	14
ResetLargestExecutionTime			11	11	11	11	11	11
GetStackOffset			21	21	21	21	21	21
Memory copy			15	15	15	15	15	15
Context save/restore			30	30	30	30	30	30
Floating point support			12	12	12	12	12	12
Interrupt support			107	107	107	107	107	107

Extended

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	163	210	253	165	214	270
	NS		200	242	284	204	250	304
	KL	2	119	166	208	121	170	226
TerminateTask	LExt	3	69	69	69	69	69	69
	H	5	91	91	91	91	91	91
ChainTask	SWL	1, 8	189	236	279	193	244	303
	SWH	1, 9	219	264	306	223	272	325
	NSL	8	237	288	331	241	298	357
	NSH	9	259	308	352	263	318	373
Schedule			140	140	159	140	140	159
GetTaskID			33	33	33	33	33	33
GetTaskState			160	160	160	162	162	162
EnableAllInterrupts			18	18	18	18	18	18
DisableAllInterrupts			31	31	31	31	31	31
ResumeAllInterrupts			43	43	43	43	43	43
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			47	47	47	47	47	47
GetResource	Task	7	262	262	231	262	262	231
	Combined	6	229	229	229	229	229	229
	CLEx	3	225	225	225	225	225	225
ReleaseResource	Task	7	233	233	233	233	233	233
	Combined	6	274	274	274	274	274	274
	CLEx	3	212	212	212	212	212	212
SetEvent	SW	1	n/a	n/a	n/a	201	201	279
	NS		n/a	n/a	n/a	237	237	315
	NS1i	10	n/a	n/a	n/a	147	n/a	n/a
	KL	2	n/a	n/a	n/a	157	157	235
	KL1i	2, 10	n/a	n/a	n/a	113	n/a	n/a
ClearEvent			n/a	n/a	n/a	82	82	82
GetEvent			n/a	n/a	n/a	113	113	113
WaitEvent	<default>		n/a	n/a	n/a	278	278	433
	fp	11	n/a	n/a	n/a	300	300	483

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	1i	10	n/a	n/a	n/a	124	n/a	n/a
GetAlarmBase			119	119	119	119	119	119
GetAlarm			122	122	122	122	122	122
SetRelAlarm			176	176	176	176	176	176
SetAbsAlarm			199	199	199	199	199	199
CancelAlarm			103	103	103	103	103	103
InitCounter			158	158	158	158	158	158
GetCounterValue			124	124	124	124	124	124
osek_tick_alarm	<default>		74	74	74	74	74	74
	KL	2	34	34	34	34	34	34
osek_incr_counter			32	32	32	32	32	32
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			125	125	125	125	125	125
ShutdownOS	NoHook	12	23	23	23	23	23	23
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			26	26	26	26	26	26
StopCOM			33	33	33	33	33	33
ReadFlag			26	26	26	26	26	26
ResetFlag			23	23	23	23	23	23
ReceiveMessage	CCCA	14	106	106	106	213	213	213
	CCCB	15	213	213	213	213	213	213
GetMessageResource			62	62	62	62	62	62
ReleaseMessageResource			62	62	62	62	62	62
GetMessageStatus			71	71	71	71	71	71
SendMessage	SW CCCA	1, 14	131	131	131	256	256	256
	SW CCCB	1, 15	235	235	235	256	256	256
	NS CCCA	14	131	131	131	256	256	256
	NS CCCB	15	235	235	235	256	256	256
	KL CCCA	2, 14	101	101	101	226	226	226
	KL CCCB	2, 15	205	205	205	226	226	226
main_dispatch	NoHook	12	161	161	190	161	161	190
	Hook	13	186	186	215	186	186	215
sub_dispatch	B1LF	19	10	10	10	10	10	10
	B1HI	20	57	57	57	57	57	57
	B1HF	21	63	63	63	63	63	63

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
	B2LI	22	n/a	28	55	n/a	28	55
	B2LF	23	n/a	34	61	n/a	34	61
	B2HI	24	n/a	81	149	n/a	81	149
	B2HF	25	n/a	87	155	n/a	87	155
	E1HI	26	n/a	n/a	n/a	256	256	330
	E1HF	27	n/a	n/a	n/a	262	262	336
	E2HI	28	n/a	n/a	n/a	n/a	n/a	330
	E2HF	29	n/a	n/a	n/a	n/a	n/a	336
ErrorHook support		16	68	68	68	68	68	68
	ServiceID	17	73	73	73	73	73	73
	Parameters	18	88	88	88	88	88	88
validity_checks		3	30	30	30	30	30	30
Timing_dispatch		4	34	34	34	34	34	34
Timing_termination		4	49	49	49	49	49	49
ActivateTaskset	SW	1	210	254	299	217	278	332
	NS		239	281	326	245	305	359
	KL	2	165	209	255	173	233	287
ChainTaskset	SWL	1, 8	258	307	353	266	329	383
	SWH	1, 9	290	350	396	298	373	427
	NSL	8	295	344	390	303	366	420
	NSH	9	321	381	427	329	404	458
GetTasksetRef			92	92	92	92	92	92
MergeTaskset			156	156	156	156	156	156
AssignTaskset			110	110	110	110	110	110
RemoveTaskset			158	158	158	158	158	158
TestSubTaskset			165	165	165	165	165	165
TestEquivalentTaskset			162	162	162	162	162	162
TickSchedule	SW	1	243	176	176	176	176	176
	NS		285	240	240	240	240	240
	KL	2	202	137	137	137	137	137
AdvanceSchedule	SW	1	254	187	187	187	187	187
	NS		296	251	251	251	251	251
	KL	2	215	150	150	150	150	150
StartSchedule			155	155	155	155	155	155
StopSchedule			111	111	111	111	111	111
GetScheduleStatus			140	140	140	140	140	140
GetScheduleValue			117	117	117	117	117	117

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			59	59	59	59	59	59
SetScheduleNext			108	108	108	108	108	108
GetArrivalpointDelay			82	82	82	82	82	82
SetArrivalpointDelay			92	92	92	92	92	92
GetArrivalpointTasksetRef			78	78	78	78	78	78
GetArrivalpointNext			82	82	82	82	82	82
SetArrivalpointNext			122	122	122	122	122	122
TestArrivalpointWritable			93	93	93	93	93	93
GetExecutionTime			84	84	84	84	84	84
GetLargestExecutionTime			72	72	72	72	72	72
ResetLargestExecutionTime			67	67	67	67	67	67
GetStackOffset			21	21	21	21	21	21
Memory copy			15	15	15	15	15	15
Context save/restore			30	30	30	30	30	30
Floating point support			12	12	12	12	12	12
Interrupt support			107	107	107	107	107	107

Notes

Number	Note
1	Linked only if upward activations are allowed
2	Linked only if API is called within ISR
3	Present only in Extended OS status
4	Present only in Timing or Extended OS status
5	Linked only if there are heavyweight tasks in the system
6	Linked only if Resource is used by both tasks and ISRs
7	Linked only if Resource is used only by tasks
8	Linked only if Chaining task is Lightweight
9	Linked only if Chaining task is Heavyweight
10	Linked only if Idle task is the only extended task in the system
11	Linked only if calling Extended task uses floating-point
12	Linked only if neither Pre- nor Post-TaskHook is used
13	Linked only if Pre- or Post-TaskHook is used
14	Linked only if there are no flags, message queues, or message resources in the system, and COM status is not requested.
15	Linked only if there are any flags, message queues, or message resources in the system, or COM status is requested.

Number	Note
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = TRUE`
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets

4.2.4 Reserved Hardware Resources

Timer units, interrupts, traps and other hardware resources are not reserved by RTA-OSEK.

4.3 Performance

4.3.1 Execution Times for RTA-OSEK API Calls

The following tables give the execution time (in CPU cycles) for each API call. (Note that: (1) the OSEK COM class was set to CCCA for systems without events and to CCCB for systems with events; (2) `ShutdownOS()` enters an infinite loop; the execution time for `ShutdownOS()` reported below is the time up to the point at which `ShutdownOS()` calls `ShutdownHook()`).

Standard

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	68	116	182	72	118	180
	NS	61	109	175	65	111	173
	KL	50	88	155	52	93	162
TerminateTask	LExt	0	0	0	0	0	0
	H	114	112	117	112	112	117
ChainTask	SWL	191	233	347	225	278	370
	SWH	245	283	395	278	328	419
	NSL	191	233	347	225	278	370
	NSH	240	278	390	273	323	414
Schedule	SW	64	66	77	66	66	77
GetTaskID		35	36	36	36	36	36
GetTaskState		74	75	74	85	84	85
EnableAllInterrupts		14	14	14	14	14	14
DisableAllInterrupts		26	26	26	26	26	26
ResumeAllInterrupts		19	19	19	19	19	19
SuspendAllInterrupts		34	34	34	34	34	34
ResumeOSInterrupts		19	19	19	19	19	19
SuspendOSInterrupts		34	34	34	34	34	34
GetResource	Task	44	44	47	47	47	50
	Combined	54	54	54	57	57	57
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	58	58	58	60	60	60
	Combined	67	67	67	70	70	70
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	98	98	98
	NS	n/a	n/a	n/a	98	98	98
	KL	n/a	n/a	n/a	76	76	77
ClearEvent		n/a	n/a	n/a	44	44	44
GetEvent		n/a	n/a	n/a	32	32	32
WaitEvent	<default>	n/a	n/a	n/a	310	299	373
	fp	n/a	n/a	n/a	320	308	383
GetAlarmBase		70	74	74	70	70	70
GetAlarm		82	85	85	82	82	82

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		98	101	101	98	98	98
SetAbsAlarm		91	94	94	91	91	91
CancelAlarm		56	57	57	56	56	56
InitCounter		63	65	65	63	63	63
GetCounterValue		62	64	64	62	62	62
osek_tick_alarm	<default>	82	83	83	82	82	82
	KL	56	57	57	56	56	56
osek_incr_counter		13	14	14	13	13	13
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		446	445	445	445	445	445
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	26	26	26	26	26	26
InitCOM		10	10	10	10	10	10
CloseCOM		10	10	10	10	10	10
StartCOM		38	38	38	210	210	210
StopCOM		28	28	28	28	28	28
ReadFlag		n/a	n/a	n/a	10	10	10
ResetFlag		n/a	n/a	n/a	7	7	7
ReceiveMessage		56	58	58	260	260	260
GetMessageResource		n/a	n/a	n/a	97	97	97
ReleaseMessageResource		n/a	n/a	n/a	101	101	101
GetMessageStatus		n/a	n/a	n/a	59	59	59
SendMessage	SW	136	194	244	331	372	447
	NS	126	184	234	327	368	443
	KL	97	146	197	295	330	413
ActivateTaskset	SW	53	352	467	59	358	479
	NS	46	345	460	52	351	472
	KL	23	328	443	29	334	455
	SW2	53	352	467	59	358	479
	NS2	46	345	460	52	351	472
	KL2	23	328	443	29	334	455
ChainTaskset	SWL	184	477	638	213	520	670
	SWH	242	530	691	270	573	723
	NSL	184	477	638	213	520	670
	NSH	237	525	686	265	568	718
GetTasksetRef		28	27	28	27	28	27

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		54	54	54	54	54	54
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		56	56	56	56	56	56
TestSubTaskset		66	66	66	66	66	66
TestEquivalentTaskset		62	62	62	62	62	62
TickSchedule	SW	143	460	568	160	478	598
	NS	130	447	555	147	465	585
	KL	117	434	542	134	452	572
	SW2	143	459	574	160	465	586
	NS2	130	446	561	147	452	573
	KL2	117	433	548	134	439	560
AdvanceSchedule	SW	110	430	538	130	448	568
	NS	97	417	525	117	435	555
	KL	82	399	507	99	417	537
	SW2	110	429	544	130	435	556
	NS2	97	416	531	117	422	543
	KL2	82	398	513	99	404	525
StartSchedule		76	76	76	76	76	76
StopSchedule		57	57	57	57	57	57
GetScheduleStatus		80	80	80	80	80	80
GetScheduleValue		66	66	66	66	66	66
GetScheduleNext		29	29	29	29	29	29
SetScheduleNext		35	35	35	35	35	35
GetArrivalpointDelay		26	26	26	26	26	26
SetArrivalpointDelay		30	30	30	30	30	30
GetArrivalpointTasksetRef		22	22	22	22	22	22
GetArrivalpointNext		26	26	26	26	26	26
SetArrivalpointNext		30	30	30	30	30	30
TestArrivalpointWritable		32	32	32	32	32	32
GetExecutionTime		11	11	11	11	11	11
GetLargestExecutionTime		25	24	24	24	24	24
ResetLargestExecutionTime		14	13	13	13	13	13
GetStackOffset		30	30	30	30	30	30

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	70	119	185	74	122	185
	NS	63	112	178	67	115	178
	KL	52	91	158	54	97	167
TerminateTask	LExt	0	0	0	0	0	0
	H	248	244	249	244	244	249
ChainTask	SWL	358	398	514	393	449	542
	SWH	406	442	556	440	493	585
	NSL	358	398	514	393	449	542
	NSH	401	437	551	435	488	580
Schedule	SW	66	68	80	68	68	80
GetTaskID		35	36	36	36	36	36
GetTaskState		75	76	75	87	86	87
EnableAllInterrupts		14	14	14	14	14	14
DisableAllInterrupts		26	26	26	26	26	26
ResumeAllInterrupts		19	19	19	19	19	19
SuspendAllInterrupts		34	34	34	34	34	34
ResumeOSInterrupts		19	19	19	19	19	19
SuspendOSInterrupts		34	34	34	34	34	34
GetResource	Task	44	44	47	47	47	50
	Combined	54	54	54	57	57	57
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	58	58	58	60	60	60
	Combined	67	67	67	70	70	70
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	98	98	98
	NS	n/a	n/a	n/a	98	98	98
	KL	n/a	n/a	n/a	76	76	77
ClearEvent		n/a	n/a	n/a	44	44	44
GetEvent		n/a	n/a	n/a	32	32	32
WaitEvent	<default>	n/a	n/a	n/a	442	429	506
	fp	n/a	n/a	n/a	452	438	516
GetAlarmBase		70	74	74	70	70	70
GetAlarm		82	85	85	82	82	82

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		98	101	101	98	98	98
SetAbsAlarm		91	94	94	91	91	91
CancelAlarm		56	57	57	56	56	56
InitCounter		63	65	65	63	63	63
GetCounterValue		62	64	64	62	62	62
osek_tick_alarm	<default>	82	83	83	82	82	82
	KL	56	57	57	56	56	56
osek_incr_counter		13	14	14	13	13	13
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		924	923	923	922	922	922
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	26	26	26	26	26	26
InitCOM		10	10	10	10	10	10
CloseCOM		10	10	10	10	10	10
StartCOM		38	38	38	210	210	210
StopCOM		28	28	28	28	28	28
ReadFlag		n/a	n/a	n/a	10	10	10
ResetFlag		n/a	n/a	n/a	7	7	7
ReceiveMessage		56	58	58	260	260	260
GetMessageResource		n/a	n/a	n/a	98	98	98
ReleaseMessageResource		n/a	n/a	n/a	101	101	101
GetMessageStatus		n/a	n/a	n/a	59	59	59
SendMessage	SW	138	197	247	333	376	452
	NS	128	187	237	329	372	448
	KL	99	149	200	297	334	418
ActivateTaskset	SW	53	352	467	59	358	480
	NS	46	345	460	52	351	473
	KL	23	328	443	29	334	456
	SW2	53	352	467	59	358	480
	NS2	46	345	460	52	351	473
	KL2	23	328	443	29	334	456
ChainTaskset	SWL	350	640	803	380	688	839
	SWH	401	686	849	430	734	885
	NSL	350	640	803	380	688	839
	NSH	396	681	844	425	729	880
GetTasksetRef		28	27	28	27	28	27

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		54	54	54	54	54	54
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		56	56	56	56	56	56
TestSubTaskset		66	66	66	66	66	66
TestEquivalentTaskset		62	62	62	62	62	62
TickSchedule	SW	143	460	568	160	478	598
	NS	130	447	555	147	465	585
	KL	117	434	542	134	452	572
	SW2	143	459	574	160	465	587
	NS2	130	446	561	147	452	574
	KL2	117	433	548	134	439	561
AdvanceSchedule	SW	110	430	538	130	448	568
	NS	97	417	525	117	435	555
	KL	82	399	507	99	417	537
	SW2	110	429	544	130	435	557
	NS2	97	416	531	117	422	544
	KL2	82	398	513	99	404	526
StartSchedule		76	76	76	76	76	76
StopSchedule		57	57	57	57	57	57
GetScheduleStatus		80	80	80	80	80	80
GetScheduleValue		66	66	66	66	66	66
GetScheduleNext		29	29	29	29	29	29
SetScheduleNext		35	35	35	35	35	35
GetArrivalpointDelay		26	26	26	26	26	26
SetArrivalpointDelay		30	30	30	30	30	30
GetArrivalpointTasksetRef		22	22	22	22	22	22
GetArrivalpointNext		26	26	26	26	26	26
SetArrivalpointNext		30	30	30	30	30	30
TestArrivalpointWritable		32	32	32	32	32	32
GetExecutionTime		76	76	76	75	75	75
GetLargestExecutionTime		36	34	34	34	34	34
ResetLargestExecutionTime		25	23	23	23	23	23
GetStackOffset		30	30	30	30	30	30

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	307	336	402	294	339	405
	NS	344	372	439	332	376	438
	KL	285	315	381	273	318	384
TerminateTask	LExt	280	275	280	275	275	280
	H	314	308	313	308	308	313
ChainTask	SWL	646	671	786	666	722	814
	SWH	693	714	828	712	765	857
	NSL	689	711	827	706	763	854
	NSH	731	749	864	747	801	892
Schedule	SW	102	105	117	105	105	117
GetTaskID		43	44	44	44	44	44
GetTaskState		308	320	319	324	323	324
EnableAllInterrupts		22	22	22	22	22	22
DisableAllInterrupts		34	34	34	34	34	34
ResumeAllInterrupts		33	33	33	33	33	33
SuspendAllInterrupts		42	42	42	42	42	42
ResumeOSInterrupts		33	33	33	33	33	33
SuspendOSInterrupts		42	42	42	42	42	42
GetResource	Task	567	594	295	673	670	363
	Combined	257	257	257	321	321	321
	CLEx	303	307	307	370	370	370
ReleaseResource	Task	285	285	285	352	352	352
	Combined	267	267	267	332	332	332
	CLEx	271	275	275	337	337	337
SetEvent	SW	n/a	n/a	n/a	337	337	337
	NS	n/a	n/a	n/a	365	365	365
	KL	n/a	n/a	n/a	323	323	323
ClearEvent		n/a	n/a	n/a	66	66	66
GetEvent		n/a	n/a	n/a	288	288	288
WaitEvent	<default>	n/a	n/a	n/a	497	483	553
	fp	n/a	n/a	n/a	507	492	563
GetAlarmBase		228	239	239	228	228	228
GetAlarm		240	250	250	240	240	240

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		288	302	302	288	288	288
SetAbsAlarm		275	288	288	275	275	275
CancelAlarm		214	222	222	214	214	214
InitCounter		389	418	418	389	389	389
GetCounterValue		204	212	212	204	204	204
osek_tick_alarm	<default>	91	92	92	91	91	91
	KL	56	57	57	56	56	56
osek_incr_counter		13	14	14	13	13	13
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		964	963	963	962	962	962
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	30	30	30	30	30	30
InitCOM		10	10	10	10	10	10
CloseCOM		10	10	10	10	10	10
StartCOM		50	50	50	222	222	222
StopCOM		39	39	39	39	39	39
ReadFlag		n/a	n/a	n/a	43	43	43
ResetFlag		n/a	n/a	n/a	32	32	32
ReceiveMessage		172	178	178	376	376	376
GetMessageResource		n/a	n/a	n/a	501	501	501
ReleaseMessageResource		n/a	n/a	n/a	495	495	495
GetMessageStatus		n/a	n/a	n/a	157	157	157
SendMessage	SW	480	544	594	679	719	798
	NS	512	580	629	723	760	837
	KL	440	505	555	643	683	762
ActivateTaskset	SW	474	830	929	505	832	947
	NS	501	856	955	547	858	973
	KL	434	794	907	483	796	925
	SW2	474	830	929	505	832	947
	NS2	501	856	955	547	858	973
	KL2	434	794	907	483	796	925
ChainTaskset	SWL	838	1189	1350	917	1239	1397
	SWH	888	1235	1396	966	1285	1443
	NSL	869	1219	1380	947	1269	1427
	NSH	914	1260	1421	991	1310	1468
GetTasksetRef		262	250	251	250	251	250

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		109	109	109	109	109	109
AssignTaskset		80	80	80	80	80	80
RemoveTaskset		111	111	111	111	111	111
TestSubTaskset		119	119	119	119	119	119
TestEquivalentTaskset		115	115	115	115	115	115
TickSchedule	SW	230	956	1060	643	983	1107
	NS	264	990	1094	677	1017	1141
	KL	200	928	1032	615	955	1079
	SW2	230	954	1067	643	956	1085
	NS2	264	988	1101	677	990	1119
	KL2	200	926	1039	615	928	1057
AdvanceSchedule	SW	201	938	1042	625	965	1089
	NS	241	971	1075	658	998	1122
	KL	180	913	1017	600	940	1064
	SW2	201	936	1049	625	938	1067
	NS2	241	969	1082	658	971	1100
	KL2	180	911	1024	600	913	1042
StartSchedule		129	129	129	129	129	129
StopSchedule		84	84	84	84	84	84
GetScheduleStatus		106	106	106	106	106	106
GetScheduleValue		92	92	92	92	92	92
GetScheduleNext		53	53	53	53	53	53
SetScheduleNext		84	84	84	84	84	84
GetArrivalpointDelay		61	61	61	61	61	61
SetArrivalpointDelay		68	68	68	68	68	68
GetArrivalpointTasksetRef		51	51	51	51	51	51
GetArrivalpointNext		55	55	55	55	55	55
SetArrivalpointNext		92	92	92	92	92	92
TestArrivalpointWritable		61	61	61	61	61	61
GetExecutionTime		93	93	93	92	92	92
GetLargestExecutionTime		250	238	238	238	238	238
ResetLargestExecutionTime		239	227	227	227	227	227
GetStackOffset		30	30	30	30	30	30

4.3.2 OS Start-up Time

OS start-up time is the time from the entry to the `StartOS()` function to the execution of the first instruction in a user task (including the idle task) without any hook routines being called. This time is always application dependent, since `StartOS()` may activate any number of tasks and start any number of user-specified alarms.

4.3.3 Interrupt Latencies

Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function. The following tables give the interrupt latencies (in CPU cycles).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	19	19	19	19	19	19
	Cat 2	19	19	19	19	19	19

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	19	19	19	19	19	19
	Cat 2	124	124	124	124	124	124

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	19	19	19	19	19	19
	Cat 2	124	124	124	124	124	124

4.3.4 Task Switching Times

Task switching time is the time between the last instruction of the previous task and the first instruction of the next task. The switching time differs, depending on the switching contexts (e.g. an `ActivateTask()` versus a `ChainTask()`).

RTA-OSEK sub-task types also affect the switching time. The tables in this section show the switching times (in CPU cycles) for all system classes for basic, lightweight tasks and for basic and extended heavyweight tasks.

Figures 1 to 8 show the RTA-OSEK switching contexts measured.

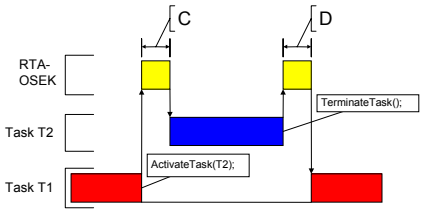

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

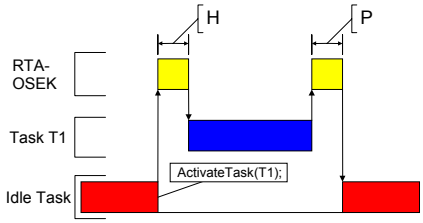

Figure 3: Task Activation from Idle Task

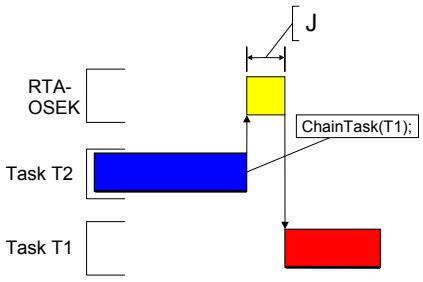

Figure 2: Task Chaining

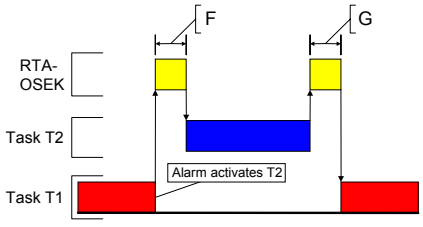

Figure 4: Task Activation from an Alarm

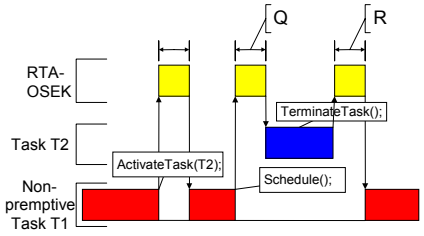

Figure 5: Non-Premptive Task Calls Schedule()

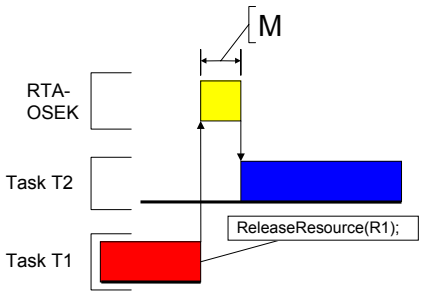

Figure 6: Blocked Task Activated by ReleaseResource()

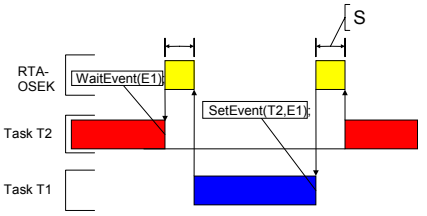

Figure 7: Waiting Task Activated by SetEvent()

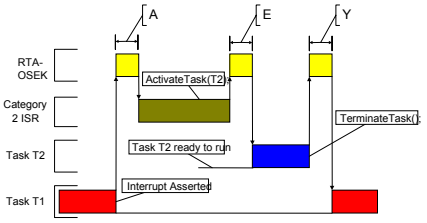

Figure 8: Category 2 ISR Activates a Higher Priority Task

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	63	86	122	63	84	122
Figure 1: D	Heavy, Basic/Extended	113	125	169	147	143	188
ChainTask	Light, Basic	144	205	308	149	208	327
Figure 2: J	Heavy, Basic/Extended	330	399	544	369	421	584
Pre-emption	Light, Basic	129	193	285	134	186	304
Figure 1: C	Heavy, Basic/Extended	174	217	333	218	252	378
From idle task	Light, Basic	129	193	285	134	186	304
Figure 3: H	Heavy, Basic/Extended	174	217	333	218	252	378
Triggered by alarm	Light, Basic	221	284	376	226	278	396
Figure 4: F	Heavy, Basic/Extended	266	308	424	310	344	470
Schedule	Light, Basic	111	120	173	111	118	174
Figure 5: Q	Heavy, Basic/Extended	158	152	214	197	189	252
Release resource	Light, Basic	128	138	179	128	136	180
Figure 6: M	Heavy, Basic/Extended	175	170	220	214	207	258
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	375	362	520

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
From category 2 ISR	Light, Basic	95	105	146	95	103	147
Figure 8: E	Heavy, Basic/Extended	142	137	187	181	174	225

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	202	220	254	202	216	254
Figure 1: D	Heavy, Basic/Extended	248	253	299	278	272	319
ChainTask	Light, Basic	318	371	475	323	376	497
Figure 2: J	Heavy, Basic/Extended	633	687	835	668	712	879
Pre-emption	Light, Basic	230	287	378	235	279	397
Figure 1: C	Heavy, Basic/Extended	271	314	432	319	354	483
From idle task	Light, Basic	230	287	378	235	279	397
Figure 3: H	Heavy, Basic/Extended	271	314	432	319	354	483
Triggered by alarm	Light, Basic	322	378	469	327	371	489
Figure 4: F	Heavy, Basic/Extended	363	405	523	411	446	575
Schedule	Light, Basic	212	212	264	212	209	265
Figure 5: Q	Heavy, Basic/Extended	253	244	308	296	285	350
Release resource	Light, Basic	229	232	272	229	229	273
Figure 6: M	Heavy, Basic/Extended	270	264	316	313	305	358
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	462	448	611
From category 2 ISR	Light, Basic	338	339	379	338	338	382
Figure 8: E	Heavy, Basic/Extended	379	371	423	422	414	467

Extended

Configuration Events Shared Task Priorities Multiple Task Activations	Task Attributes	Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Normal termination	Light, Basic	280	297	330	280	293	331
Figure 1: D	Heavy, Basic/Extended	314	317	364	344	337	385
ChainTask	Light, Basic	605	643	746	607	660	780
Figure 2: J	Heavy, Basic/Extended	985	1022	1171	1017	1060	1228
Pre-emption	Light, Basic	463	500	591	461	502	623
Figure 1: C	Heavy, Basic/Extended	505	528	646	546	578	710
From idle task	Light, Basic	463	500	591	461	502	623
Figure 3: H	Heavy, Basic/Extended	505	528	646	546	578	710
Triggered by alarm	Light, Basic	564	600	691	562	603	724
Figure 4: F	Heavy, Basic/Extended	606	628	746	647	679	811
Schedule	Light, Basic	244	243	295	244	240	296
Figure 5: Q	Heavy, Basic/Extended	286	276	340	329	317	382
Release resource	Light, Basic	451	454	494	502	502	546
Figure 6: M	Heavy, Basic/Extended	493	487	539	587	579	632
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	700	686	849
From category 2 ISR	Light, Basic	355	356	396	355	355	399
Figure 8: E	Heavy, Basic/Extended	397	389	441	440	432	485

4.4 Configuration of Run-time Context

The run-time contexts of all tasks reside on the same stack and are recovered when the task terminates. As a result, run-time contexts of mutually exclusive tasks are effectively overlaid. The RTA-OSEK GUI is able to calculate the worst-case stack requirement for the entire application, based on the declared stack usage, the priorities and the resource occupation of individual tasks.

The size of the run-time context of a task depends on the task type and the system configuration. The following tables give the sizes (in bytes) for different OS status and configurations:

Standard

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	18	18	20	18	18	20
BCC1 lightweight, floating-point	20	20	22	20	20	22
BCC1 heavyweight, integer	25	25	27	25	25	27
BCC1 heavyweight, floating-point	25	25	27	25	25	27
BCC2 lightweight, integer	n/a	20	24	n/a	20	24
BCC2 lightweight, floating-point	n/a	20	24	n/a	20	24
BCC2 heavyweight, integer	n/a	25	29	n/a	25	29
BCC2 heavyweight, floating-point	n/a	25	29	n/a	25	29
ECC1 heavyweight, integer	n/a	n/a	n/a	27	27	29
ECC1 heavyweight, floating-point	n/a	n/a	n/a	27	27	29
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	29
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	29
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	20	20	20	20	20	20
BCC1 lightweight, floating-point	22	22	22	22	22	22
BCC1 heavyweight, integer	27	27	27	27	27	27
BCC1 heavyweight, floating-point	27	27	27	27	27	27
BCC2 lightweight, integer	n/a	22	24	n/a	22	24
BCC2 lightweight, floating-point	n/a	22	24	n/a	22	24
BCC2 heavyweight, integer	n/a	27	29	n/a	27	29
BCC2 heavyweight, floating-point	n/a	27	29	n/a	27	29
ECC1 heavyweight, integer	n/a	n/a	n/a	29	29	29
ECC1 heavyweight, floating-point	n/a	n/a	n/a	29	29	29
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	29
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	29

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		26	26	28	26	26	28
BCC1 lightweight, floating-point		28	28	30	28	28	30
BCC1 heavyweight, integer		33	33	35	33	33	35
BCC1 heavyweight, floating-point		33	33	35	33	33	35
BCC2 lightweight, integer		n/a	28	32	n/a	28	32
BCC2 lightweight, floating-point		n/a	28	32	n/a	28	32
BCC2 heavyweight, integer		n/a	33	37	n/a	33	37
BCC2 heavyweight, floating-point		n/a	33	37	n/a	33	37
ECC1 heavyweight, integer		n/a	n/a	n/a	35	35	37
ECC1 heavyweight, floating-point		n/a	n/a	n/a	35	35	37
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	37
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	37
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		28	28	28	28	28	28
BCC1 lightweight, floating-point		30	30	30	30	30	30
BCC1 heavyweight, integer		35	35	35	35	35	35
BCC1 heavyweight, floating-point		35	35	35	35	35	35
BCC2 lightweight, integer		n/a	30	32	n/a	30	32
BCC2 lightweight, floating-point		n/a	30	32	n/a	30	32
BCC2 heavyweight, integer		n/a	35	37	n/a	35	37
BCC2 heavyweight, floating-point		n/a	35	37	n/a	35	37
ECC1 heavyweight, integer		n/a	n/a	n/a	37	37	37
ECC1 heavyweight, floating-point		n/a	n/a	n/a	37	37	37
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	37
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	37

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		26	26	28	26	26	28
BCC1 lightweight, floating-point		28	28	30	28	28	30
BCC1 heavyweight, integer		33	33	35	33	33	35
BCC1 heavyweight, floating-point		33	33	35	33	33	35
BCC2 lightweight, integer		n/a	28	32	n/a	28	32
BCC2 lightweight, floating-point		n/a	28	32	n/a	28	32
BCC2 heavyweight, integer		n/a	33	37	n/a	33	37
BCC2 heavyweight, floating-point		n/a	33	37	n/a	33	37
ECC1 heavyweight, integer		n/a	n/a	n/a	35	35	37
ECC1 heavyweight, floating-point		n/a	n/a	n/a	35	35	37
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	37
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	37
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		28	28	28	28	28	28
BCC1 lightweight, floating-point		30	30	30	30	30	30
BCC1 heavyweight, integer		35	35	35	35	35	35
BCC1 heavyweight, floating-point		35	35	35	35	35	35
BCC2 lightweight, integer		n/a	30	32	n/a	30	32
BCC2 lightweight, floating-point		n/a	30	32	n/a	30	32
BCC2 heavyweight, integer		n/a	35	37	n/a	35	37
BCC2 heavyweight, floating-point		n/a	35	37	n/a	35	37
ECC1 heavyweight, integer		n/a	n/a	n/a	37	37	37
ECC1 heavyweight, floating-point		n/a	n/a	n/a	37	37	37
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	37
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	37

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.